

FIG. 1

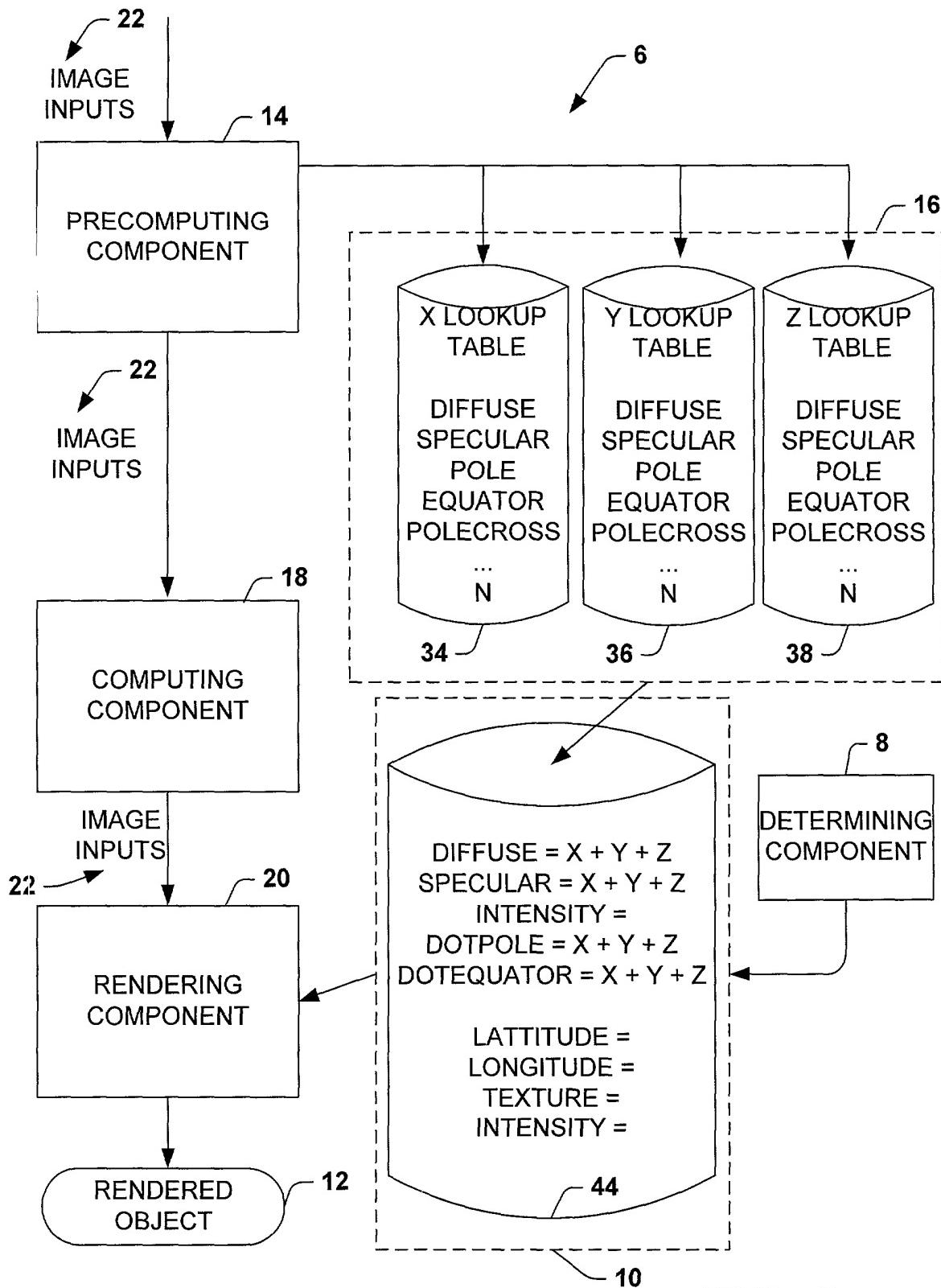


FIG. 2

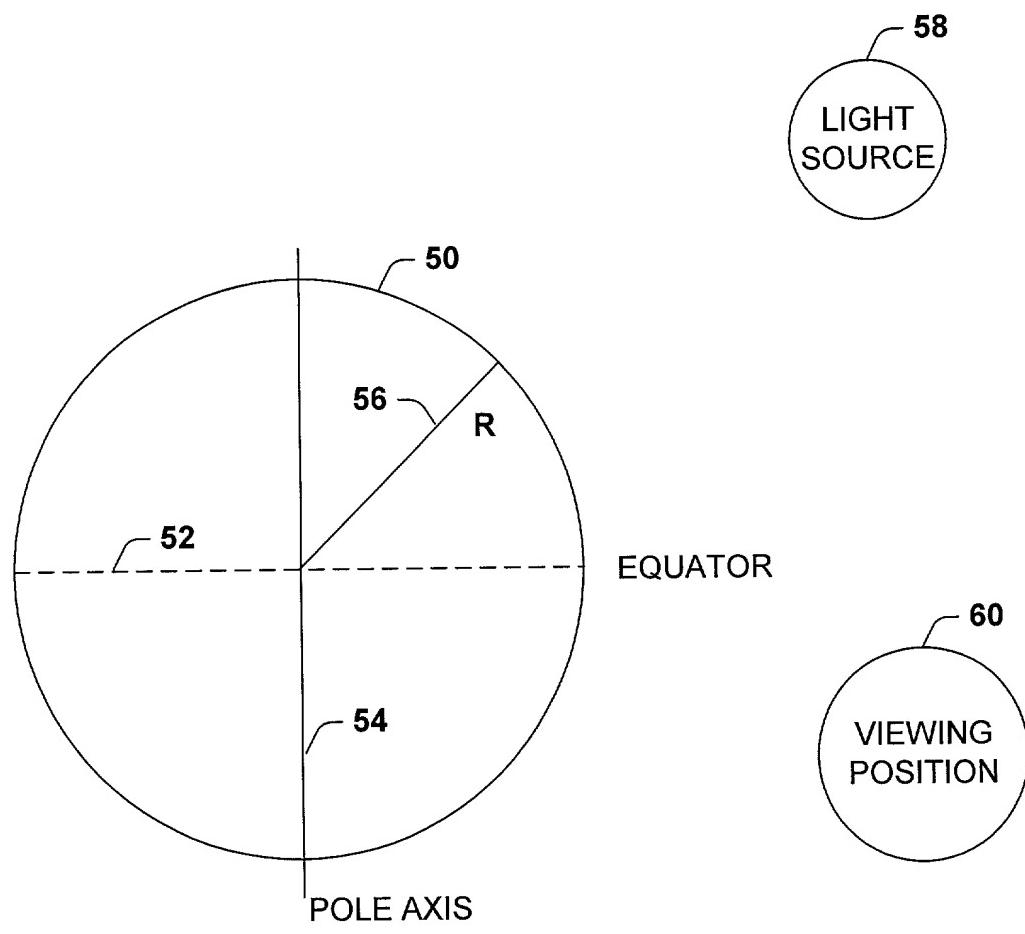


FIG. 3

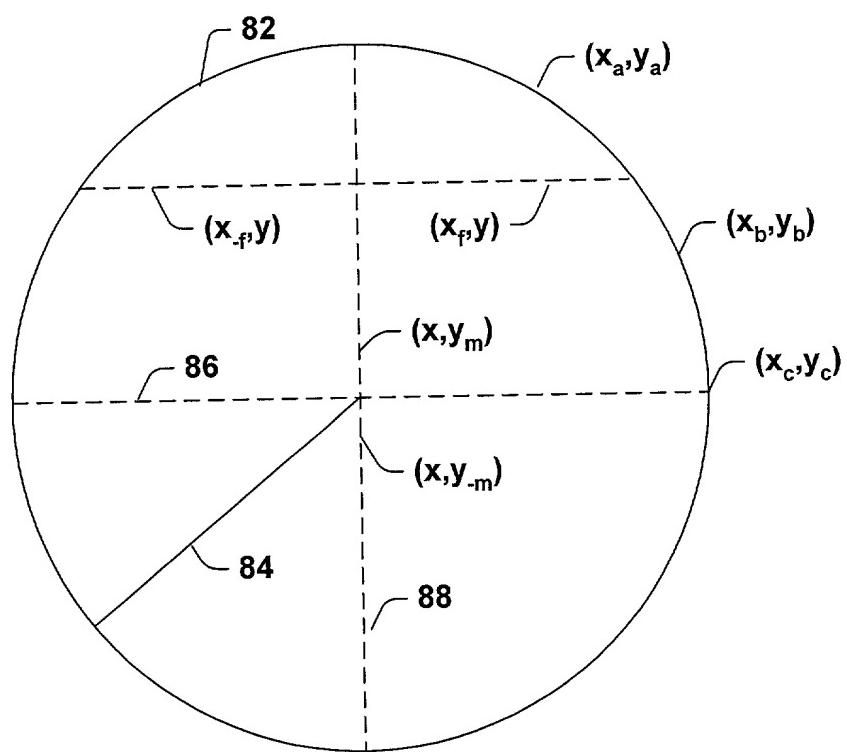
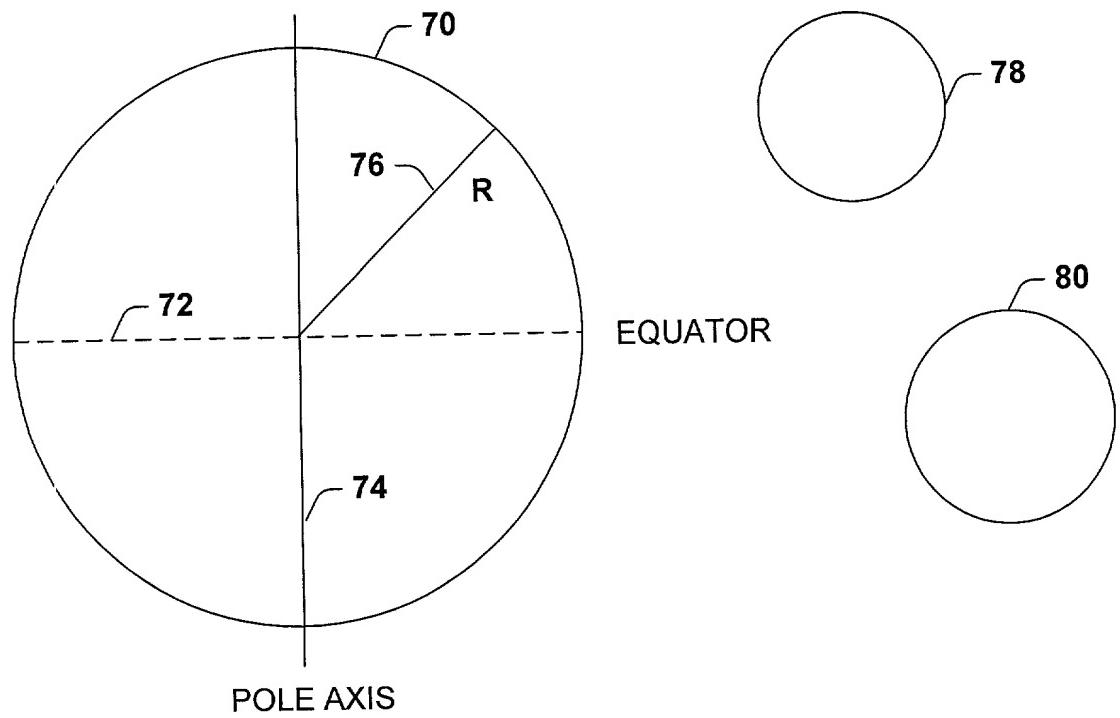


FIG. 4

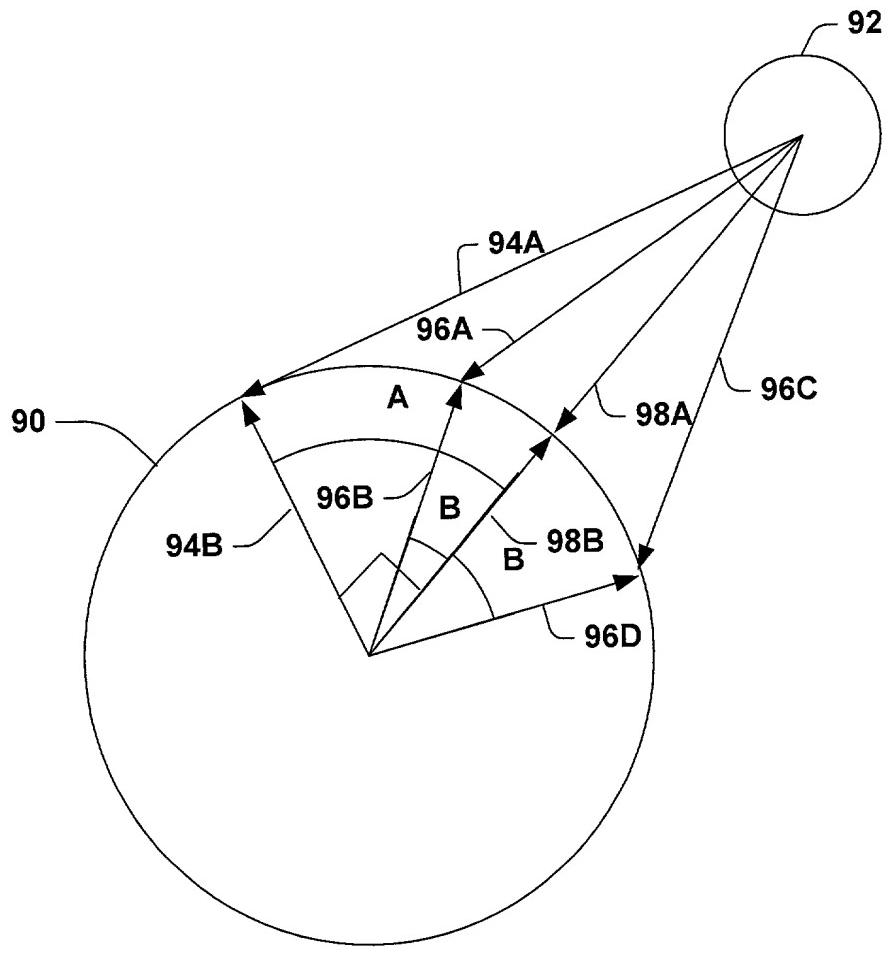


FIG. 5

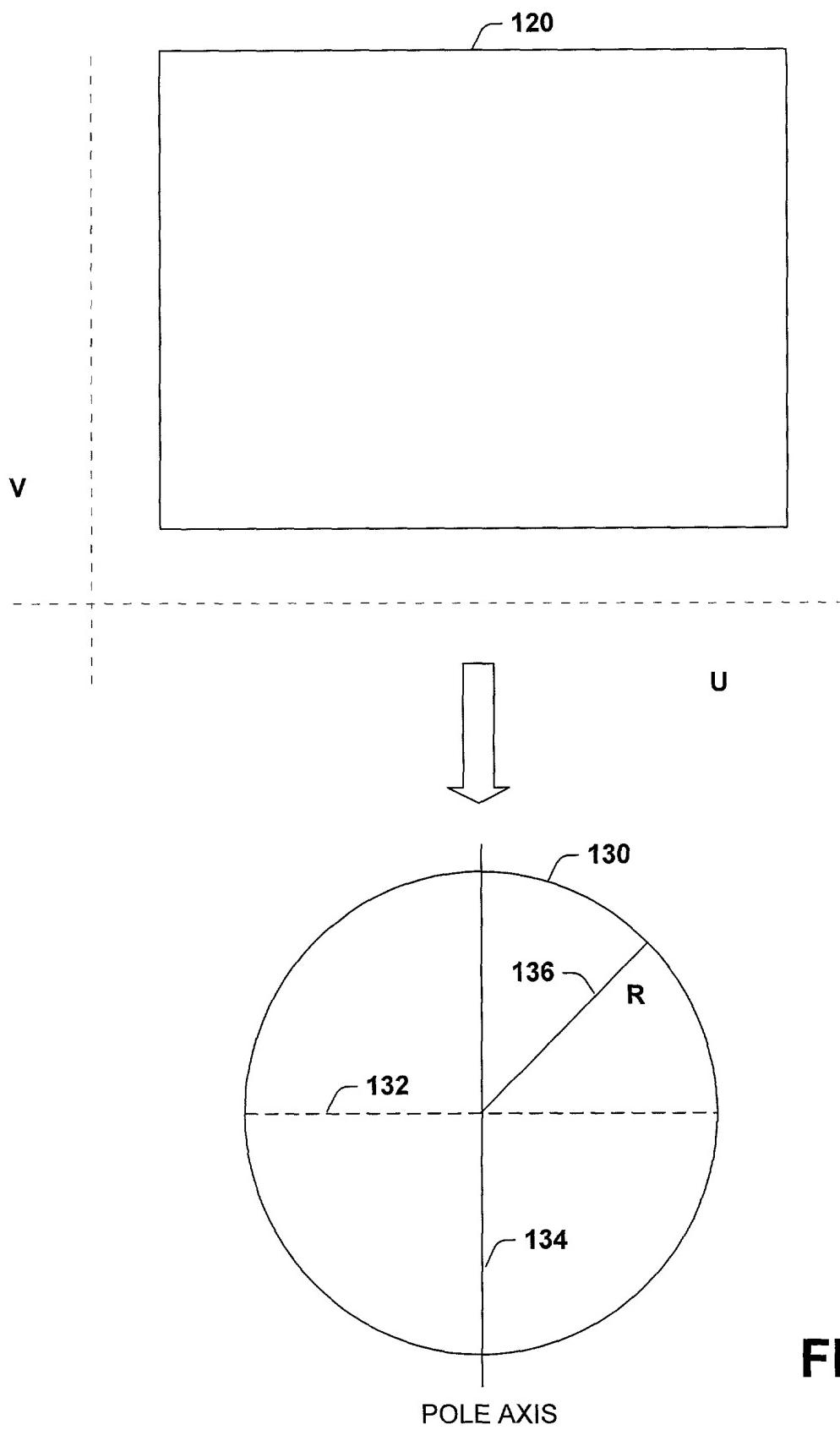


FIG 6

```

Draw A Sphere (input Radius, CenterX, CenterY, VectLight, VectViewer,
              VectorPole, VectorEquator)
    // the image inputs include the size of the sphere, where it is to be drawn,
    // where a lighting source is positioned and where a viewer is positioned
{
    // set up initial vectors
    vectSpecularHighlight = Normalize(vectViewer + vectLight);
    vectPoleCrossEquator = VectorPole cross VectorEquator;

    // prepare lookup tables, can be computed before rendering
    // portions of later calculations pre-calculated here b/c x & y invariant to
    // other parameters
    for ( i = -rad; i <= rad; i++)
    {
        j = i * 1 / rad;
        xMultiplyDiffuse[i] = j * vectLight.x;; // setup diffuse component
        yMultiplyDiffuse[i] = j * vectLight.y;
        xMultiplySpecular[i] = j * vectSpecularHighlight.x; // setup specular
        yMultiplySpecular[i] = j * vectSpecularHighlight.y;
        xMultiplyPole_LUT[i] = j * vectorPole.x; // used for texture
        yMultiplyPole_LUT[i] = j * vectorPole.y;
        xMultiplyEquator_LUT[i] = j * vectorEquator.x; // setup equator
        yMultiplyEquator_LUT[i] = j * vectorEquator.y;
        xMultiplyPE_LUT[i] = j * vectPoleCrossEquator.x; // where pole &
        xMultiplyPE_LUT[i] = j * vectPoleCrossEquator.y; //equator cross
    }
    for ( x = 0; x < rad; x++) // finite set of discriminants
    {
        disc = r^2 - x^2;
        for ( y = 0; y < x; y++) // thus finite set of z values
        {
            disc = disc - y^2;
            if ( disc > 0 )
            {
                zInvRad = 1 / (squareroot(disc) * radius;
                zMultiplyDiffuse_LUT[disc] = zInvRad * vectLight.z;
                zMultiplySpecular_LUT[disc] = zInvRad *
                    vectSpecularHighlight.z;
                zMultiplyPole_LUT[disc] = zInvRad * vectorPole.z;
                zMultiplyEquator_LUT[disc] = zInvRad * vectorEquator.z;
                zMultiplyPE_LUT[disc] = zInvRad *
                    vectPoleCrossEquator.z;
            } // end if
        } // end for y
    } // end for x
// proc cont'd on Fig. 7b

```

FIG 7A

```

// Iterate over the scanlines in the sphere
// combining the precomputed lookup elements as you go
// for each scan line
for ( y = -rad; y <= rad; y++ )
{
    RadiusSubYSquare = r^2 - y^2;
    Bound = edgeBuffer[abs(y)]; // bound is the horizontal displacement from
                                // y axis
    for ( x = (-bound + 1); x <= bound; x++ )
    {
        152 // iterate over every pixel in the scanline y
        disc = RadiusSubYSquare - x^2; // compute disc for look up table
                                        // index
        diffuse = yMultiplyDiffuse[y] + xMultiplyDiffuse[x] +
                  zMultiplyDiffuse_LUT[disc];
        specular = yMultiplySpecular[y] + xMultiplySpecular[x] +
                  zMultiplyDiffuse_LUT[disc];
        specular = SpecularRemapLUT[specular]; // remap to range 0 -1.0

        150 // compute the final intensity for a pixel
        intensity = diffuse * diffuseFactor + specular * specularFactor +
                    ambient * ambientFactor;
        // compute the u & v texture components for a pixel
        NormalDotPole = xMultiplyPole_LUT[x] + yMultiplyPole_LUT[y] +
                        zMultiplyPole_LUT[z];
        NormalDotEquator = xMultiplyEquator_LUT[x] +
                           yMultiplyEquator_LUT[y] + zMultiplyEquator_LUT[z];
        latitude = arccos(NormalDotPole);
        vTexture = latitude/PI;
        longitude' = NormalDotEquator / sine(latitude);
        clamp longitude' to range -1.0 to 1.0
        longitude = arccos(longitude');

        // determine how longitude wraps around sphere
        if (xMultiplyPE_LUT[x] + yMultiplyPE_LUT[y] +
            zMultiplyPE_LUT[disc] < 0 )
        {
            uTexture = longitude;
        }
        else
        {
            uTexture = 1 - longitude;
        }

        // fetch a textured pixel from coordinate uTexture, vTexture
        // scale intensity of textured pixel by Intensity
        // draw the lit, texture pixel at location ( x + centerX, y + centerY)
    } // end for x
} // end for y
} // end proc

```

FIG. 7B

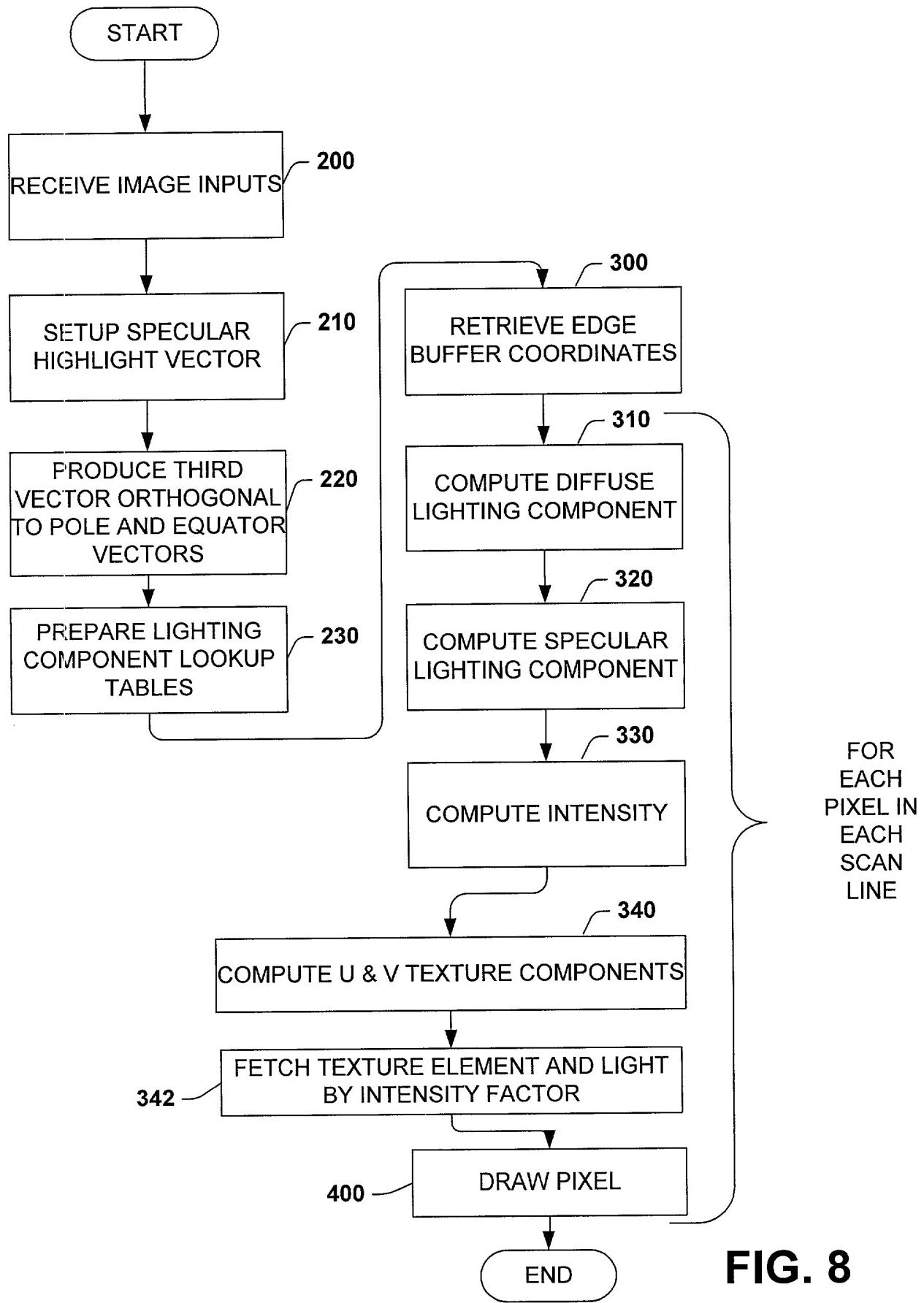


FIG. 8

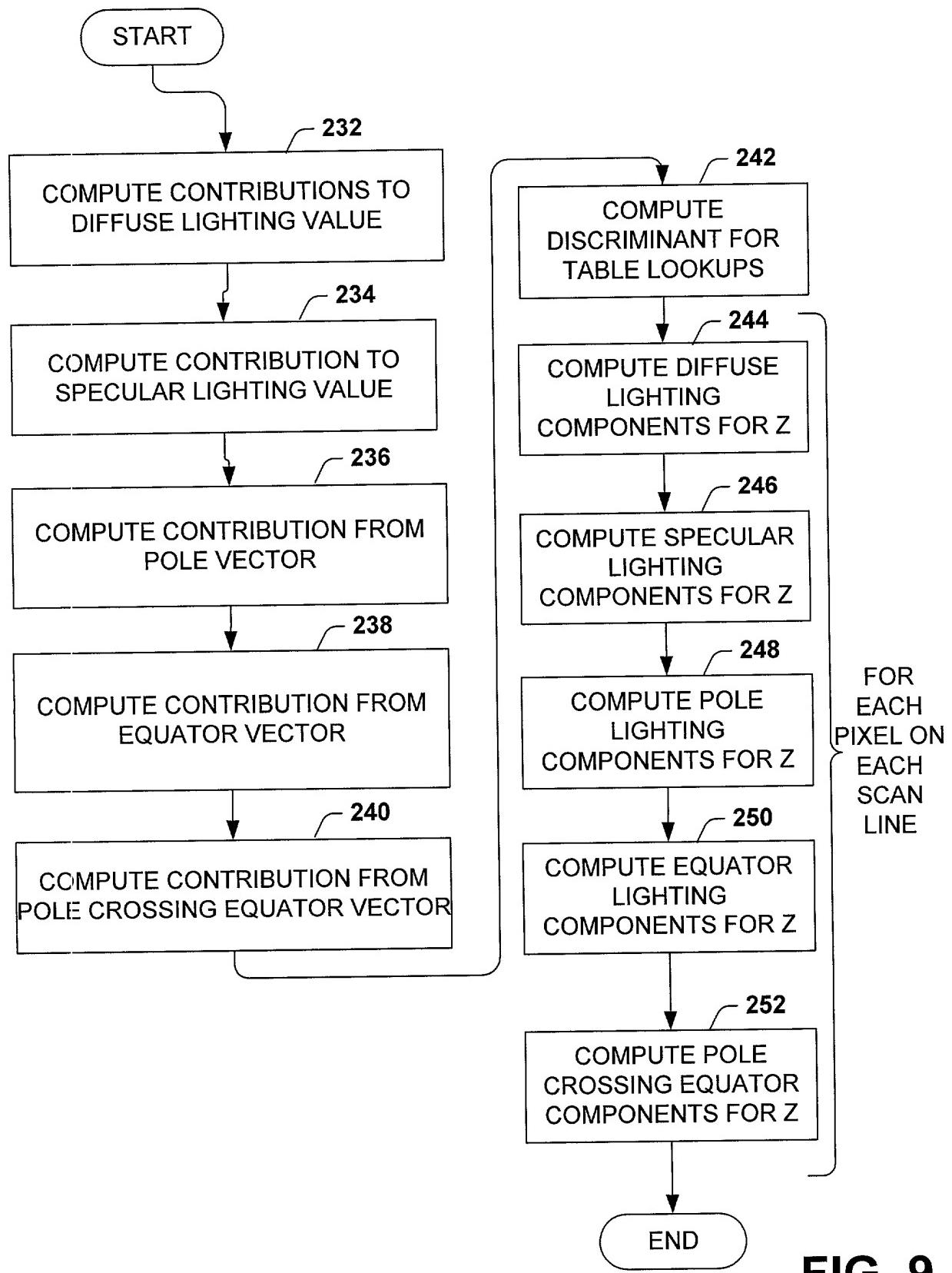
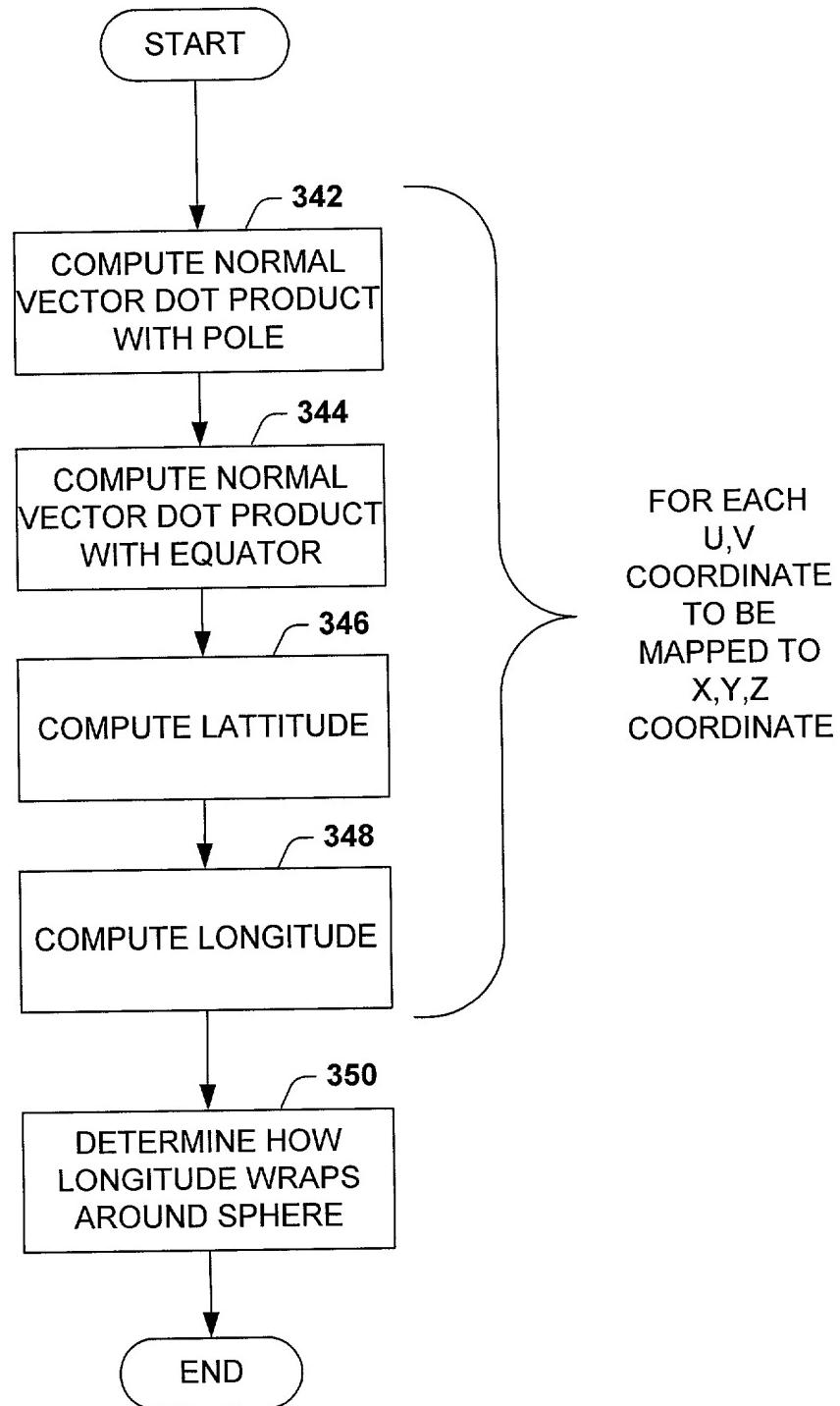


FIG. 9

**FIG. 10**

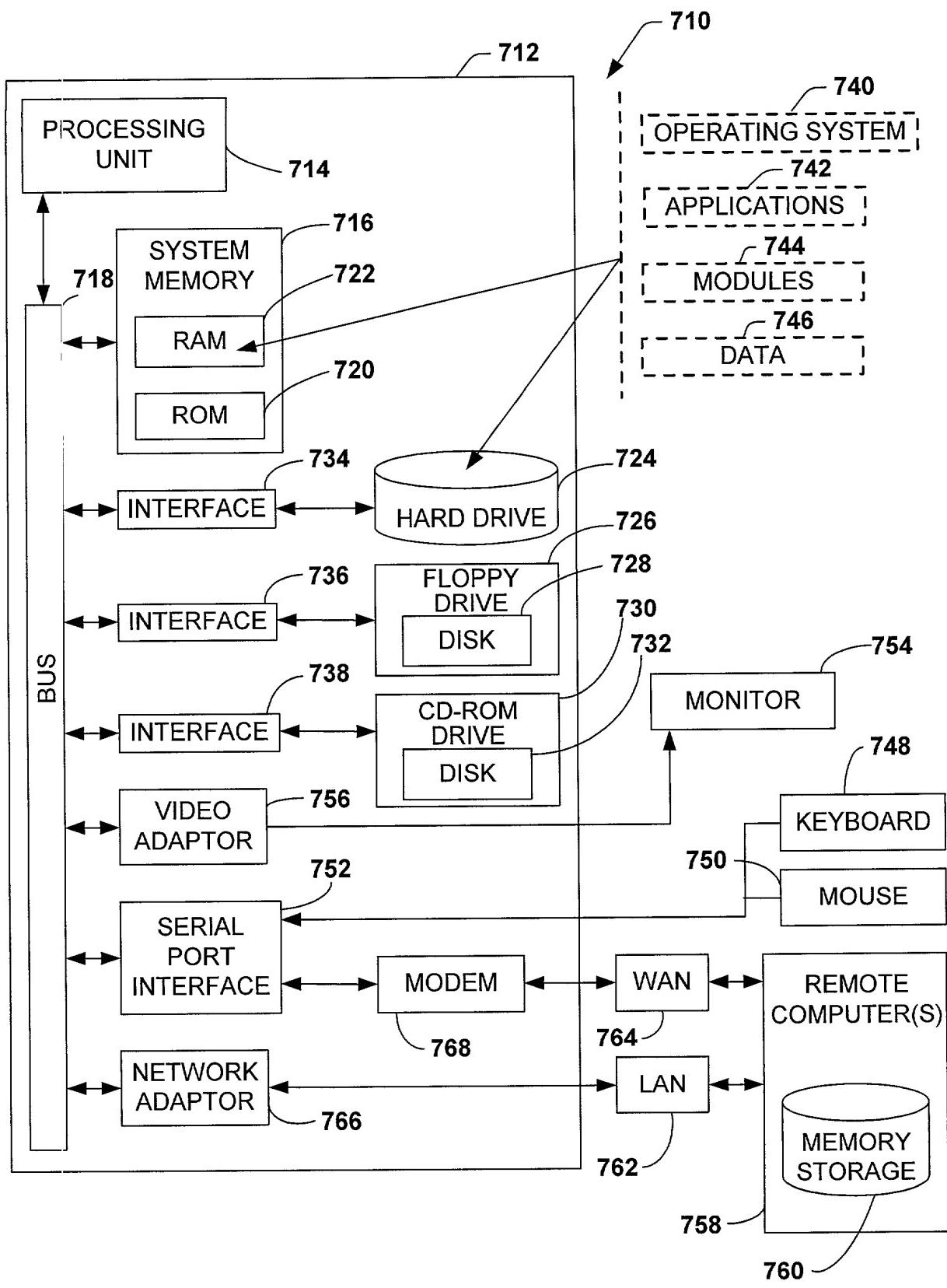


FIG. 11